Master's Programme in New Media

Full Body Problem!

Animation control for virtual reality characters

Jukka Huiskonen

Aalto University

MASTER'S
THESIS

Author
Jukka Huiskonen

Title
Full Body Problem! Animation control for virtual reality characters

School School of Arts, Design and Architecture

Master's programme New Media

Major Game Design and Production

Supervisor professor Perttu Hämäläinen

Advisor professor Perttu Hämäläinen

Level Master's thesis **Date** 16 Apr 2018 **Pages** 31 **Language** English

Abstract

This thesis aims to produce a solution which can create motion for full body virtual characters from the input of a typical virtual reality device and illustrates the issues when mapping the input motion from a virtual reality device to a full body character, an issue which is titled the 'Full Body Problem.'

The technical part of the thesis is created using Unity3D, a game engine. The animations used as a data source for the solution were a combination of motion capture animations created in Aalto MotionLab and premade clips from Unity Technologies.

The written part explores the theoretical background for animating virtual characters and controlling them in real-time applications using various animation control methods. The definition of the Full Body Problem is introduced and discussed. An overview of the different technical parts is presented before an evaluation using a side-by-side comparison between the user motion and the motion created by the solution.

The result of this thesis is a technical solution which produces relatively natural-looking motion of a virtual character from the user's motion. The solution is controlled by a typical virtual reality device. The results provide insights for researchers and developers looking to develop similar systems. In conclusion, using a hybrid approach of direct mapping and animation control to solve the problem of limited tracking data is a solid basis for solving the Full Body Problem.

Keywords animation, data-driven animation, virtual reality

Tekijä
Jukka Huiskonen

Työn nimi
Kokonaisen kehon ongelma! Tiedon ohjaamat hahmot virtuaalitodellisuudessa

Korkeakoulu Taiteiden ja suunnittelun korkeakoulu

Maisteriohjelma Uusi media

Pääaine Pelituotanto ja suunnittelu

Valvoja professori Perttu Hämäläinen

Ohjaaja professori Perttu Hämäläinen

Työn laji Opinnäyte **Päiväys** 16.04.2018 **Sivuja** 31 **Kieli** englanti

Tiivistelmä

Tämän opinnäytetyön tavoitteena on tuottaa tekninen ratkaisu jolla kyetään ohjaamaan virtuaalisia hahmoja tyypillisen virtuaalitodellisuuslaitteen syötteellä. Tavoitteena on myös tuoda esille haasteita virtuaalisten hahmojen ohjauksessa.

Opinnäytetyön tekninen osuus on luotu käyttäen pelimoottori Unity3D:tä. Ratkaisussa käytettyjen animaatioiden lähteenä on hyödynnetty Unity Technologies:n tarjoamia animaatioita, sekä Aalto-yliopiston MotionLab:ssä tuotettuja liikekaappausanimaatiota.

Kirjallisessa osuudessa tarkastellaan virtuaalihahmojen animoinnin teoriataustaa ja niiden ohjaamista reaaliaikaisissa ohjelmissa. Osuudessä esitellään myös kokonaisen kehon ongelma sekä tehdään katsaus ratkaisun tekniseen osuuteen. Lopuksi arvioidaan ratkaisun kyky tuottaa luonnollisen näköistä liikettä vertaamalla rinnakkain käyttäjän ja ratkaisun tuottamia liikkeitä.

Opinnäytetyön tuloksena on tekninen ratkaisu, joka pystyy tuottamaan suhteellisen luonnollisen näköistä liikettä virtuaalitodellisuuslaitteista saadun syötteen perusteella. Ratkaisu antaa monia oivalluksia tutkijoille ja kehittäjille, jotka ovat kehittämässä samankaltaisia ratkaisuja. Lopputuloksena hybridimalli, jossa käyttäjän syötteen suoranainen käyttö yhdistettynä animaatiokontrolliin osoittautui toimivaksi ratkaisuksi tilanteessa, missä käyttäjän syöte ei sisällä tietoa koko kehon liikkeistä.

Avainsanat animaatio, tiedon ohjaama animaatio, virtuaalitodellisuus

Aalto University
MASTER'S THESIS 2018

Full Body Problem!

Animation control for virtual reality characters

Jukka Huiskonen

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Arts.
Otaniemi, 16 Apr 2018

Supervisor: professor Perttu Hämäläinen
Advisor: professor Perttu Hämäläinen

Aalto University
School of Arts, Design and Architecture
Master's Programme in New Media

Contents

List of Figures and Tables

1. Introduction

Naturally moving virtual characters have been the goal of real-time application from their inception. As the visuals for these characters begin to edge closer to photorealistic, so must the motion they express reflect the same level of fidelity.

Motion capture has made it easier to produce natural looking animations in situations where the motion of the virtual character is under scrutiny (Parent, 2012). Though the controls for these character used in real-time applications, such as games, have remained relatively similar for the past decade.

Although the concept of virtual reality (VR) has existed from the mid-20th century and VR devices have been used in research for the past couple of decades, the major jump from being a niche field of study to a multi-billion-dollar industry happened relatively recently. (Bailenson, 2018)

The release of devices such as HTC Vive, Oculus Rift (Figure 1.1), and PlayStation VR has brought the idea of using VR devices into the mainstream. A change, which has not gone unnoticed in a wide range of industries.

These VR devices raise an interesting question; can we replicate the user's motion with a virtual character using just the data from the VR devices? Trying to replicate the user's motion is problematic because most VR devices do not provide tracking for the full body. Without the use of additional equipment, the untracked motion must be interpreted from the tracking data. I have titled this as the 'Full Body Problem'.

During the search for a suitable topic for my thesis, professor Perttu Hämäläinen suggested that I investigate data-driven animations. This suggestion fit my interests well. After a while, prof. Perttu Hämäläinen arranged a meeting with the developers at Fake Production, a company

Figure 1.1. Oculus Rift head-mounted display (HMD) and Oculus Touch controllers.

which recently began developing a multi-user collaborative VR platform. Their target is to create a platform where natural communication between VR users is facilitated by expressive virtual characters (Nikula, 2017).

During my work in Fake Production, the thesis grew from simple overview and examination of data-driven animations to a solution which produces motion for full body virtual characters controlled by a VR device. The research question turned out to be; can a solution, which uses data-driven animation techniques, produce natural motion for full body characters from the tracking data provided by a typical VR device?

This thesis is organized in the following manner; Chapter 2 starts with a short introduction to computer animations proceeded by an overview of different animation control methods. The Chapter 3 takes a closer look at the Full Body Problem and explores the different aspects of it, including proposed solutions. Chapter 4 explores and evaluates the solution. The Chapter 5 concludes the thesis with discussion and reflections and provides ideas for further research.

2. Theoretical background

Creating realistic and naturally moving virtual characters is a difficult problem. Real-time applications which afford controls over these characters, the problem is even more complicated as the resulting motion should look natural while still being responsive to external control (Van Welbergen, Van Basten, Egges, Ruttkay, & Overmars, 2010).

This chapter provides a concise overview of the theoretical background of this thesis. The first section takes a look at how virtual characters are animated. While the second part examines the problem of animation control and explores some of the methods used to address it.

2.1 Computer animation

Computer animations, a subset of computer graphics, deal with animating virtual objects. (Parent, 2012).

2.1.1 Character model

In computer graphics, characters are often represented using a polygonal model. In the model, character's skin is a collection of vertices and polygons. A vertex (pl. vertices) represents a single point in space, while a polygon is constructed from three interconnected vertices. These vertices can be used to create multiple polygons (Figure 2.1). (Parent, 2012)

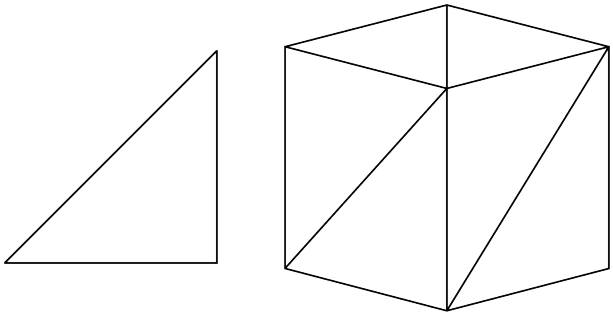

Figure 2.1. Visualization of a single polygon and a simple model made out of multiple polygons.

Computer animations work by moving the model's vertices. In three dimensional (3D) models, a single vertex might be connected to multiple faces (Figure 2.1). In the simple model, the vertices are located in the corners. Thus moving a single vertex affects all faces connected to it.

Figure 2.2 shows a wireframe visualization for a 3D character. It contains roughly 160 thousand polygons and 100 thousand vertices. Creating animations for this model by moving each vertex at a time is cumbersome and inefficient. Different high-level control methods are used to mitigate this issue. One particular solution is to use bones.

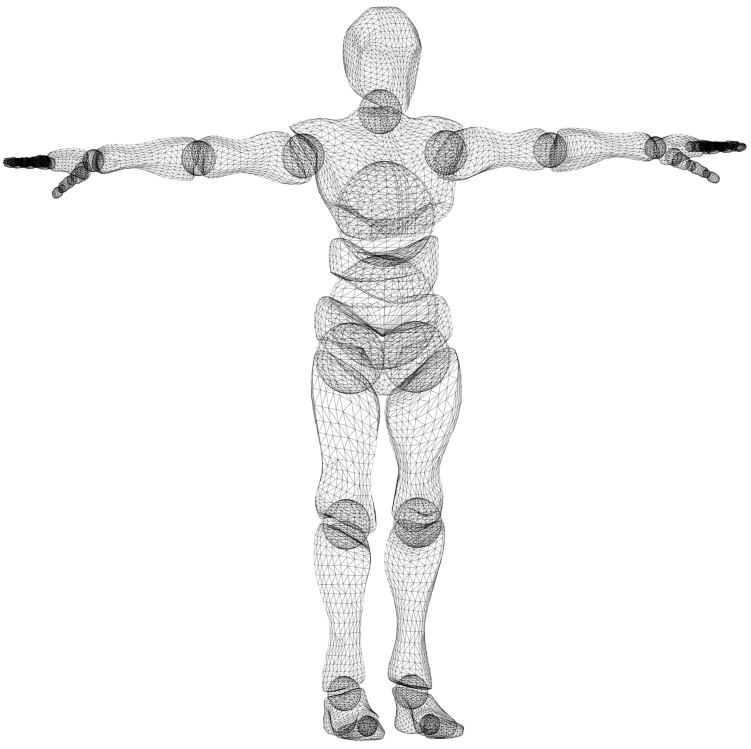

Figure 2.2. Wireframe visualization of a character model. Model from Mixamo.com (Mixamo.com, 2018).

2.1.2 Skeletal model

In skeletal animations, bones are a high-level representation for a set of vertices, while a skeleton is a hierarchical structure of bones. In this hierarchy, each bone inherits the position and rotation from its parent bone. (Van Welbergen et al., 2010)

Using skeletal model simplifies the animation data substantially, as the motion is expressed only as the local position and rotation of each bone, rather than the position of each vertex.

Skinning

A technique called skinning is used to create a connection between the bones and the vertices. The vertices are assigned a weight for each bone, which dictates how much of the bones transformation affects the vertex. (James & Twigg, 2005)

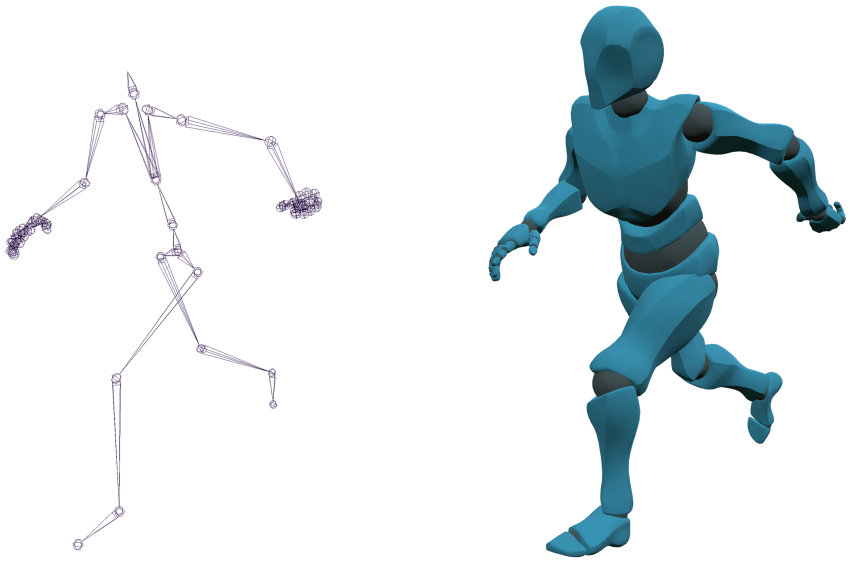

Figure 2.3. Characters skeleton and the 3D model deformed by it. Model from Mixamo.com (Mixamo.com, 2018).

The skeletal model (Figure 2.3) provides a convenient high-level method for manipulating the virtual character.

2.1.3 Key-frame animations

In key-frame animations, the local positions and rotations are stored in bones in the skeletal model. When enough of these key-frames are played one after the other, it creates the illusion of movement. However, animating a character only with the data from the key-frames would result in the

animation to behave like stop-motion animation. This motion is due to there being no intermediate values between two key-frames unless the values are created using interpolation. (Parent, 2012)

Interpolation in animations is the process of creating intermediate values between two or more key-frames. Different interpolation methods exist which are used to smooth the motion between key-frames. (Parent, 2012)

A variety of different ways of creating key-frames exists, from posing to motion capture. In hand-crafted animations, animators pose the characters skeletal model and create the key-frames from these poses. Hand-crafting realistic and convincing looking human motion is time-consuming and requires a significant amount of skill. Which is one reason motion capture has grown in popularity, as a more convenient way to create natural human key-frame animations. (Parent, 2012)

2.1.4 Motion Capture

Motion capture is the method of recording motion of real-life objects or actors.

There are multiple different methods for capturing real-life motion, one of which is called optical motion capture. This method uses markers placed all over the actor or the object. The markers are coated with a reflective material, which the infra-red (IR) sensitive cameras are calibrated to pick up. As a single camera produce a two-dimensional (2D) picture where these markers appear as simple dots, a multi-camera setup is used to capture these markers from multiple different angles. Combining the position of the marker from each camera allows the motion capture system to determine their position in 3D space. The raw data from these optical motion capture systems is a point cloud, which is not usable at first but must be first transformed to skeletal model. (Figure 2.4). (Herda, Fua, Plänkers, Boulic, & Thalmann, 2001)

Using optical motion capture has its limitations. Issues such as marker occlusions or tag mismatch can occur fairly easily. Also, these require additional manual post-processing to clean up. However, even with the burden of additional post-processing, the resulting animations provide the most physically accurate motion more efficiently than hand-crafting the animations. (Herda et al., 2001)

Whereas the previously discussed methods to produce key-frame data were 'offline' methods (i.e., created before use in the target application), procedural animations take a different approach.

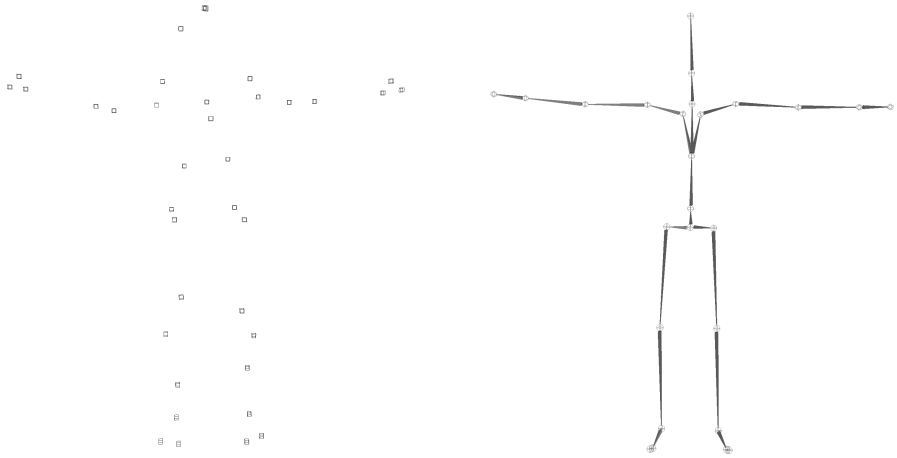

Figure 2.4. Left: Marker data capture from actor, Right: Skeleton created from the maker labeled marker data

2.1.5 Procedural animations

In procedural animations, the position and rotation for the vertices or bones are derived using an underlying mechanism, such as physics simulation. Though key-frame animations can be created using these procedural methods offline, their advantage comes from the fact that the motion is derived 'online' (i.e., while the target application is running). (Parent, 2012)

Inverse Kinematics

Inverse Kinematics (IK) is a way to achieve the desired position and rotation for a hierarchy of joints by using kinematic equations. IK was first used in robotics, where the correct angles for a series of mechanical joints with different degrees-of-freedom (DoF) were needed for the robot to reach a specific target. (Aristidou, Lasenby, Chrysanthou, & Shamir, 2017)

IK is often used as a complementary method in real-time applications together with key-frame animations. Example of a situation where IK proves useful is when a character is tasked to grab an external object. Using offline animations would require creating individual animation clips to account for each unique position of the external object, which is both tedious and time-consuming. By allowing the IK to control the hand to grab the target object, a reasonable result can be achieved without the use of extra key-frame animations.

Though there are a variety of methods to solve the situation above, such as making sure that the object is always in the correct position relative to the character. However, IK is commonly used in real-time applications

where the animations must adapt dynamically different circumstances. More in-depth review of the different IK techniques and their use in computer graphics can be found in the article by Aristidou et al. (Aristidou et al., 2017).

The previous sections provided an outline of how computer animations work, which is just the first half of the problem as real-time applications often require some way to control the animations.

2.2 Animation control

Animation control, especially related to human characters is a difficult problem as they must ensure natural motion for the character while proving responsive controls over it. (Van Welbergen et al., 2010)

This summary presents few animation control methods used in real-time applications.

2.2.1 Naive animation control

The most simple and naive animation control method is to create a direct link between specific user input and a corresponding animation. While this would produce responsive controls over the character, it would become cumbersome to develop and maintain, as the number of animations and input parameters increased.

Another problem with this approach relates to the rise of unnatural transitions. Without any way to manage which transitions between different animations are valid, the only way to prevent unnatural motions from occurring is to use animations with similar enough motion. This method works in a limited number of situations, mostly when the naturalness of the motion is not a priority or the number of animations or input parameters are sufficiently limited.

2.2.2 State machines

One of the key problems with the simple method is the lack of rules to indicating what transitions were valid. With state machines, the rules for valid transition are encoded within states and the predefined transitions between them.

A state represents a single action, such as walking or running, which has a specific animation associated with it. By explicitly defining the

Figure 2.5. Example of a simple state machine.

transitions between these states, the rules of motion emerge. (Hofkamp, 2015)

Figure 2.5 represents a simple state machine with three states including the transitions between them. The state machine in question expresses that the character must first 'walk' before transitioning to the 'run' state. Additional rules can be placed in the transitions which allow for more nuanced control of the rules of motion.

Though state machines provide an improvement over naive animation controls, they are not without flaws. As new states are added to support a broader range of motions, the number of possible transitions between these states also increases. Increasing the number of states leads to the increase in complexity, which can become problematic in applications where the characters have a significant motion space.

2.2.3 Motion Matching

Motion Matching is a data-driven animation control method developed by Ubisoft Montreal and was used in Ubisoft's action game titled For Honor (2017) (Clavet, 2016a). Motion Matching is a data-driven method, which means that rather than explicitly specifying the rules of motion, like with the state machines, the rules are generated using the data within the animations. Most data-driven control methods use motion capture animations, due to the availability of high-quality motion capture data (Holden, Komura, & Saito, 2017).

The motivation for developing Motion Matching was explained by Simon Clavet, an animation programmer at Ubisoft, in his AIIDE presentation (Clavet, 2016b). He mentioned the need for an animation control system which can produce natural motion using a large number animations, remain responsive to user input and not suffer from the increased complexity as previously seen with the state machines. In the presentation, Clavet explains that they researched multiple different data-driven techniques, but were discouraged by their complexity. (Clavet, 2016b)

Though the central idea of data-driven animations where compelling as he goes on to explain that they ended up using an idea from a paper

titled 'Motion Fields for Interactive Character Animation' (van de Panne, 2014). The idea was that the animation system could transition from one animation to another at any point in time. Though the question how to make the decision when and where to transition was still open, as Clavet mentions that they decided not to use the methods detailed paper. (Clavet, 2016b)

In Motion Matching the decision which animation to play is based on the summed cost from two separate cost-functions; pose and trajectory matching. These cost functions rely on the pose structure described below. (Clavet, 2016b)

Pose

Data-driven methods use some form of higher-dimensional representation for the animations, such as a Motion Graph (Kovar, Gleicher, & Pighin, 2008). Data-driven methods use this representational data to make decisions about which animations to play. In Motion Matching, this high-dimensional representation is called a pose (Clavet, 2016b)

Pose represents a simplified snapshot of an animation frame. Clavet presents an example of the pose structure used in For Honor (Table 2.1).

Root bone velocity
Feet bone positions relative to the root bone
Feet bone velocities
Weapon bone position

Table 2.1. Table of pose structure used in For Honor (Clavet, 2016a).

Poses are generated in a preprocessing phase, in which a 'pose library' is created from all animations. As each pose is a representation of an animation frame, the interval to create these poses affects how large the resulting pose library becomes. Clavet notes that there is no need to generate a pose per animation frame. In For Honor, they generated a pose for every 0.1s of animation, which seemed work well enough for the pose and trajectory matching while keeping the pose library sufficiently small for their performance requirements. (Clavet, 2016b)

Pose Matching

The first cost function performed in Motion Matching is called pose matching. Pose matching calculates the cost between two poses, the current and the candidate pose. The current pose is determined by keeping check of the character's current animation and at what point it is playing during the

current frame. The candidate pose represents an arbitrary pose selected from the generated poses. The method for determining the cost is a simple distance function between these two poses. If the poses are identical, the cost is zero, but as the poses become more different, e.g., the feet positions differ, the cost rises. Clavet reveals that the trick is to limit the number of bones used in the pose matching (Table 2.1). (Clavet, 2016b)

Trajectory Matching

As pose matching calculates how closely the candidate pose resembles character's current situation, trajectory matching is used to respond to the user input. Trajectory matching compares the trajectory extrapolate from the user input to one extrapolated from the candidate pose. The trajectory from the candidate pose is created by simulating where the character would end up if the candidate pose. (Clavet, 2016b)

During each frame, a goal is generated based on the user input. The goal contains a set of trajectory points. Each trajectory point has a position and orientation, which are compared to the trajectory from the candidate pose. If the trajectory cost between the user input and the candidate pose is zero, then playing the animation the candidate pose was created from would move the character precisely along the trajectory created from the user input. (Clavet, 2016b)

After calculating the cost for each pose in the pose library, the candidate pose with the lowest cost is selected. (Clavet, 2016b)

3. Full Body Problem

As mentioned in the previous chapter, creating natural looking motion in real-time applications is usually a balancing act between the naturalness of the motion and the control exerted over it. This chapter defines the Full Body Problem and explores some pre-existing solutions for it.

3.1 Definition

The Full Body Problem is a situation where there is a need to simulate full-body motion based inadequate set of input parameters. Games are one of the most pronounced applications where this problem presents itself daily.

Though the scope of this problem is directly tied to the size of the input and motion spaces. Input space refers to all possible input variations, while the motion space indicates all the possible motions a virtual character can perform. Games represent real-time applications where the input space is relatively small and well defined while the size motion space can vary.

Quality of a solution to Full Body Problem can be evaluated the resulting motion produced by the solution. A high-quality solution is responsive while producing natural-looking motion.

3.2 Solutions

In this section, I focus on the solutions for the Full Body Problem when the input space supports VR devices. Though these solutions do not refer to the Full Body Problem, they still use a similar problem definition.

3.2.1 Full Body Inverse Kinematics

As discussed in the previous chapter, inverse kinematics is a procedural way to create motion, which does not require any pre-made animations. A solution which uses inverse kinematics to determine the motion for the full body character is called full body inverse kinematics (FBIK).

In FBIK the input from VR devices is used as IK target. Though the problem with FBIK is the same as with other procedural animation methods. Even though they offer a significant amount of control, the resulting motion does not look natural and creating a mathematical models which produce in natural motion from the limited data from the VR device is a difficult task.

3.2.2 Character Modification

One of the simple solutions is to modify the virtual character to fit the tracking data. In these solutions, when using a typical VR device with three tracking points, the character appears as a disjointed head and hands floating in the air. Though this produces the most accurate representation of the user's motion, the character is no longer presented in the full body form.

3.2.3 Full Body Tracking

As the problem is the lack of necessary tracking data, another simple solution is to fill the gaps in the tracking. This can be achieved by using additional hardware or even motion capture studio, though from a practical standpoint this solution is fairly limited as it requires additional investment and decreases the ease of use of the VR device.

3.2.4 Hybrid

Hybrid solutions are trying to solve the problem by creating a two-step solution, which separates the upper and lower body motion into two distinct problems. The main idea behind hybrid solutions is to map the motion from tracked bodyparts to the corresponding body parts on the virtual character. While the motion for untracked other body parts is constructed using the data from the tracked body parts. Example of this type of hybrid solution is presented in the paper by Jiang et al. In their solution, they use a combination of inverse kinematic and pre-made animations. (Jiang,

Yang, & Feng, 2016)

By splitting the problem, hybrid solutions can offer a good balance of accurate motion reconstruction for head and hands, while still produce natural lower body motion using pre-made animations.

In the next chapter, I present my solution, which uses the hybrid approach to solve the Full Body Problem.

4. Solution

This chapter begins with an overview of my solution for the Full Body Problem and concludes with its evaluation.

4.1 Overview

The following solution for the Full Body Problem uses a previously described hybrid approach, where the problem is divided into two sub-parts; the upper-body and the lower-body. This solution was created using the HTC Vive as the target input device.

The upper-body -part of the problem is solved by mapping the input from the user to the character using an IK solution. Due to the possibility of the size difference between the user and the virtual character, the input was adjusted to account for the differences in size, between the user and the target virtual character. This part is detailed in the 'Inverse Kinematic' -section 4.4.

The more exciting and challenging part of the Full Body Problem relates to the lower-body. This is because the target VR device does not provide any tracking for the lower body. The solution must be able to produce lower-body motion from the user input. Motion Matching (Chapter 2) together with motion capture animations is used as a solution for the lower-body.

Unity3D (Unity) was selected as the development platform, because of the familiarity I had with it and its selection of animation tools. One of them is the ability to retarget animations, which means that the bones in the animations are automatically scaled to fit the target character (Technologies, 2018a).

4.2 Source animations

Motion capture animations were used as the source for the animations since the solution aims to create realistic and natural motion.

Initially, a premade set of motion capture animations from Unity's Asset Store was used. The animation set was created by Unity Technologies for testing purposes. In addition to the animations, it included a virtual character (Unity Technologies, 2012), which was used as the reference character in evaluating the solution.

The original plan was to use the above-mentioned set of motion capture animations in the development and evaluation of the solution. But due to quality issues, such as missing frames, in the motion capture animations, I chose to create my set motion capture animations.

4.2.1 Motion Capture

Aalto University's Motion Lab is a multipurpose facility which has an OptiTrack motion capture solution available for students and research groups (University, 2012).

I created a plan for the motion capture sessions using resources found from mocappys.com -site which provides guides for motion capture related subjects (Mocappys.com, 2018).

The plan consisted of two separate days of shooting. These days were scheduled to be a week apart, to allow for any additional planning which might arise from the observations of the first day. The first day was planned as an introductory to the process of capturing motion and creating an initial motion capture animation set. This initial set was to test how the animation could be imported into Unity.

The second day was used to capture a predefined set of motions, which were used throughout the development of the solution and in its evaluation.

A crucial part of the success of this process was the use of a 'shot list'. The shot list contains items such as the movement descriptions for the actors, props and their use, and all other essential details required during the shooting. The shot list was especially helpful as I redid many takes when the captured motion contained some capture artifacts, such as marker occlusion.

Animations from the second day of shooting where of high-quality and did not have any noticeable artifacts. Though the previously mentioned animation retargeting functionality in Unity would have allowed the vir-

tual character to use these animations without any extra processing, I still decided to use Autodesk MotionBuilder to post-process and retarget the animations. The reason was to gain familiarity with the motion capture toolchain and create a pipeline for future use.

4.3 Motion Matching

The following Motion Matching implementation is based on the material from two presentations by Simon Clavet (Clavet, 2016b), (Clavet, 2016a). Though modifications were done to support inputs from VR devices.

The implementation is presented module by module, starting with the processing of the source animations to the poses and their use in pose matching. After this, I'll detail the problem of creating the desired trajectory from the VR device input. I conclude the section with an overview of the additional procedural touchup used in the solution.

4.3.1 Poses

As previously mentioned, Motion Matching uses poses as the underlying data structure. It uses these poses to make the decision about which animations are played. This is why the pose structure and their generation was used as the starting point for the implementation.

Though, before defining the poses or generating them, a reliable method for manipulating animations inside Unity was required. More specifically, a way to play animation clips with an arbitrary frame-rate. I received help from my thesis supervisor, prof. Perttu Hämäläinen, in the form of an example implementation, which I modified to suit the solution.

The pose structure presented in Clavet's presentation (Clavet, 2016b) was used as the initial structure for the pose structure for the solution. I did not want to explicitly define the structure (e.g., feet and arms) as it would require modifications every time I changed which bones to pose match. Thus I used a list of the bones which were utilized in pose matching. The index of the list denoted which bone was in question. This structure saved me from modifying the pose structure multiple times during testing, especially as I often changed which bones were used in the pose matching.

The pose generator is tasked to create the pose library from a set of animation clips. It does this by creating a temporary character, which is animated using Unity's animation controller to play all the animations

clips given to the pose generator.

List below shows the steps which the pose generation uses to create the pose library.

1. Play the first frame of the clip.

2. Create a new pose and calculate the root velocity (incl. angular velocity).

3. Calculate the velocities for the pose matching bones and also store their position relative to the root.

4. Advance the animation by the pose interval amount.

5. If we are at the end of the animation clip, move to next animation clip, otherwise jump to 3.

Initially, the pose generation executed every time the solution started. As an improvement, the pose library was serialized, which meant that the generation was done only when needed, though the Unity editor was still required to generate the pose library.

The pose generation is a necessary preprocessing step in Motion Matching. Implementing it first along with the pose structure proved to be a right decision, as these components contributed to my understanding of the animations as data.

Next, I'll present the two primary cost functions used in the solution, the pose, and the trajectory matching. Though the pose library generation provided the necessary data to create both cost functions, I decided to implement them separately due to the fact it simplified testing.

4.3.2 Pose Matching

As previously discussed, pose matching compares the current pose to a candidate pose and outputs a value which represents how similar the poses are.

For the pose matching, I create a mechanism to keep track which animation clip was playing and at what time. During each animation transition, the record was updated. And during each frame, the animation time was increased using the frame time.

The solution first generates a 'current pose' using the above-mentioned

animation clip and animation time. The current pose is generated by looking at the poses generated from the currently playing animation clip. The solution always uses interpolation to generate the current pose from two poses, which are closest to the current animation time.

Verifying that the current pose matched the character's animation was done by visualizing the interpolated pose's data, such as the location and orientation of the root and the position of the pose matching bones. This tool was also used to check the results from the trajectory matching.

After verifying that the current pose was generated correctly, the cost function was implemented, compared the current pose to every pose in the pose library, using a simple distance function. Though the positions and velocities functions were trivial and produced relatively small values for similar poses, the angular velocity value required some extra effort.

The pose generator stored the angular velocity in degrees with a range of [-180, 180], which caused the pose matching to favor poses with similar angular change a little too greedily. Though the problem was more pronounced in the trajectory matching, so I decided to scale the angular value to a range of [0, 1]. This scaling produced the most fitting pose and trajectory matching results.

4.3.3 Goal Generation

After verifying that the pose matching produced correct values, I started to plan on how to implement the trajectory matching. Before the trajectory matching, the desired trajectory needed to be extrapolated from the user input.

The first iteration of goal generation was done using values from a typical game controller. This was done due to the fact that the initial scope of the thesis did not contain VR devices as an input device. In retrospect, it allowed me to have a working version of a more straightforward goal generation system before tackling on the VR devices.

The goal structure took the ideas presented in Clavet's presentation, in which a goal contains a set of trajectory points. These trajectory points are extrapolated from the user input, and they contain all the information the trajectory matching needs to calculate the difference between the goal and the current candidate trajectory. (Clavet, 2016b)

Extrapolating the trajectory points from the user input is relatively straightforward, as the current delta velocity from the input is multiplied by desired speed for the character, which is then multiplied time offset. By

adding the result from the previous calculation to the character's current position, results in a position for a single trajectory point.

But as the scope of the thesis grew from using a simple gamepad controller as the input device to the Full Body Problem, the goal generator was modified to function with the input from VR devices.

The VR devices presented an interesting problem. How to transform the input data from the three different devices (HMD and two hand controllers), into goals for the trajectory matching.

I decided to use only the HMD as the data source for the goal generation and disregard the input from the controllers. My reasoning for this was the fact that movement from the head represents the movement of the body more closely than the controllers. Also selecting a single input source kept the goal generation relatively simple.

The HMD gives the user's head position and rotation in 3D. This caused problems as the trajectory points root position and orientation referred to the characters root bone, which is usually at the ground level between the feet.

To solve this issue, I took inspiration from the previously mentioned article by Jiang et al. (Jiang et al., 2016), in which they create a scaling parameter based on the height of the user and the virtual character. As my solution required a user calibration step for the IK, which is detailed later, getting the height of the user was trivial. The heigh was then used to calculate the approximate root position for the user using the HMD positional data. I used naive implementation for the root position, which places the root directly under the HMD. Though this caused some problems, using this simple approach I did not have to modify the goal generation code.

4.3.4 Trajectory Matching

The function of the trajectory matching is to produce a cost between the goal and the candidate pose. In the solution, this is done immediately after the pose matching.

In the solution, the trajectory generated from a candidate pose is done by iterating poses which are generated from the same animation clip as the candidate. The iteration sums up the root velocity for every pose and keeps track of the overall time offset by also incrementing the offset with the pose interval. When time offset goes over the trajectory point offset, a cost value is calculated between the position from trajectory point and

the point generated from the previous summing of poses. This is also the point where the difference in orientation is summed as similarly as in pose matching, by scaling the angular difference to the range of [0, 1]. This iteration is done until the cost between all the trajectory points is calculated for the candidate pose.

The previous method worked well with the game controller, but the VR input presented the following problem; even if we generated the root position from the user's head position, we would still need a way of supporting up and down movement, e.g., crouching and sitting down motion.

As the original Motion Matching system was not designed for this situation, I introduced a second position for the trajectory point, the head position. Unlike the root position, generating the head position was as simple as using the original HMD position. Goal creation was also refactored to support the head position and to extract it from the animations.

Now that the solution had both the pose and trajectory matching calculating the total cost of selecting a candidate pose, additional rules were required to instruct the solution when a transition from one animation to another was valid.

4.3.5 Animation blending

Unity offers an animation controller with which developers can define transitions between different animations and even tune the transition parameters, such as duration of the transition (Technologies, 2018b). In the solution, I tested using these transition parameters, but I noticed that there was no convenient way to transition to a specific point in time within an animation. The ability to transition to an arbitrary point within an animation was crucial because each pose represented a point in an animation clip and the solution needed some way to transition to that point.

Fortunately, Unity's animation control provided a couple of methods which fit the previously stated requirements. Initial testing resulted in less than optimal motion until advice from prof. Perttu Hämäläinen, who suggested to take the transition duration into account in the transition. At this point, I had a working motion matching system which accepted input from VR devices, though it still needed some procedural touchups.

4.3.6 Procedural Touchups

Because the solution cannot guarantee to have an animation which would fit the desired trajectory, a slight difference between the goal and the animated character is typical. In his presentations, Clavet presented methods for both correcting the character's position and rotational. These methods boil down to adding movement towards the desired position orientation to the character in addition to the movement from the animation. (Clavet, 2016b)

I experimented with various methods to manipulate the character's position, but all resulted in unrealistic motion. I had more success with the rotational correction, which improved the solutions response to turning and did not cause any noticeable motion artifacts.

One key point from Clavet's presentation was the introduction of a 'responsivity' slide, which is essentially a scaling factor for the trajectory matching, as it would be used to scale the output value from that cost function. A lower value would make the solution emphasize the pose matching over the trajectory matching and vice versa. The responsivity slider was an essential part of the solution, as it provided much-needed control over the responsiveness of the character's motion. (Clavet, 2016b)

Another interesting problem came from when the user stood still, and the solution began selecting the same idle animation in every frame, but with a slightly different time offset. To resolve this, I restricted the animation transitions when the system found a more optimal pose within the same animation. Though this solution only works when the animation clip contains only a single motion which can be looped, such as idling. A more robust system should be investigated.

4.4 Inverse Kinematics

The previously described solution alone was not adequate for reconstructing the full body motion from the user input, as it only provided an approximate animation for the lower body. An inverse kinematic solution was used to animate the hands.

As IK solutions are already used in a wide range of applications, implementing my own solution seemed unnecessary. I opted to use a premade IK solution. I searched a suitable solution from the Unity Asset Store which offers multiple free and commercial IK solutions. Though the selected IK

solution supported FBIK, I only ended up using the IK solution for head and hands.

Though the IK solution provided convenient methods for mapping the input directly from VR device to the character, this was not usable in the solution, as I wanted to avoid situations where the characters hands would be in unrealistic positions. To solve this issue, I transformed the input from the controllers to be relative to the HMD and then scaled all of them using the configuration created at the start of the solution.

4.5 Evaluation

In this section I evaluate the solution's performance, using a side-by-side comparison of the input motion and the motion produced by the solution.

I used the following a predefined list of motions in the comparison:

- Walking backwards/forward
- Turning in place
- Crouching
- Sitting

I used a video camera to capture the input motion, while the motion produced by the solution was captured using a virtual camera in combination with a screen capturing software. The video from both sources was synced and placed side-by-side using a video editing tool.

4.5.1 Character configuration

As previously mentioned, simply mapping the user input directly to the virtual character is problematic due to the difference in size between the users and the character. This why the solution requires the user to create a character configuration by first holding hands to the side and pressing triggers from both controllers and then assuming a T-pose and repeating the trigger presses (Figure 4.1).

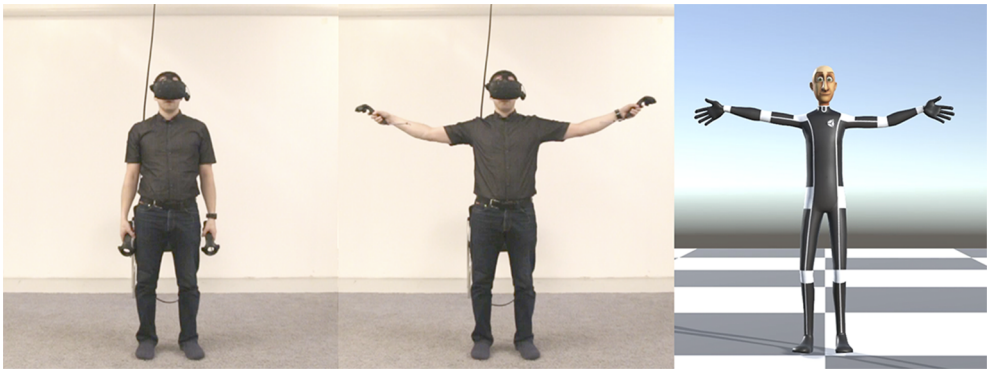

Figure 4.1. Configuring the solution and creating the character

4.5.2 Walking

The Figure 4.2 shows how the solution succeeds in modeling walking motion based on the user input.

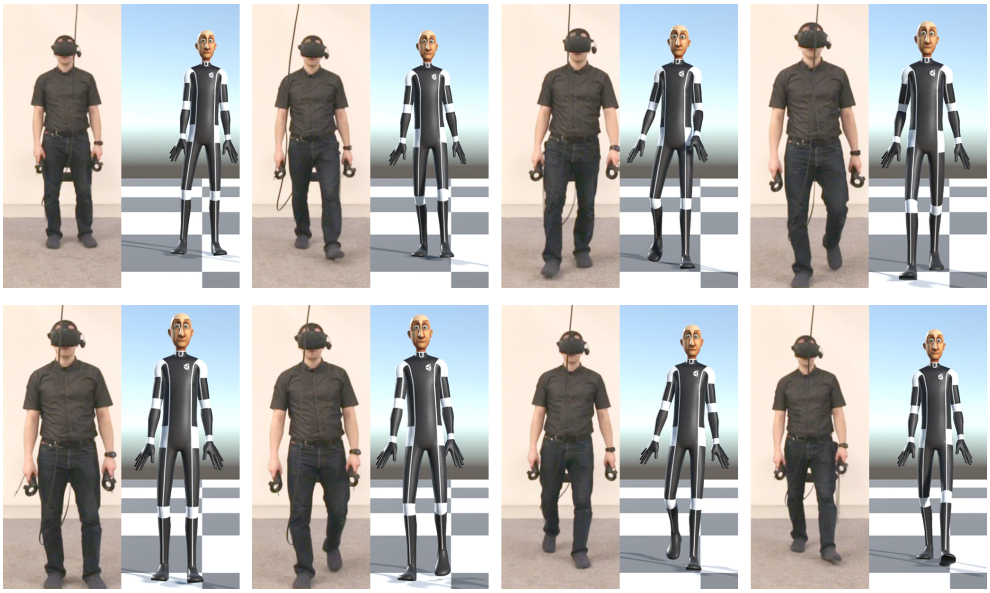

Figure 4.2. Side-by-side comparison of walking motion

The solution manages to recognize that the user is walking and the resulting motion closely resembles the user's motion. Though this is partly due to the fact that the user in question was also responsible for the source animations. Although the figure shows that both user's and the character's feet are in sync, it is not by design.

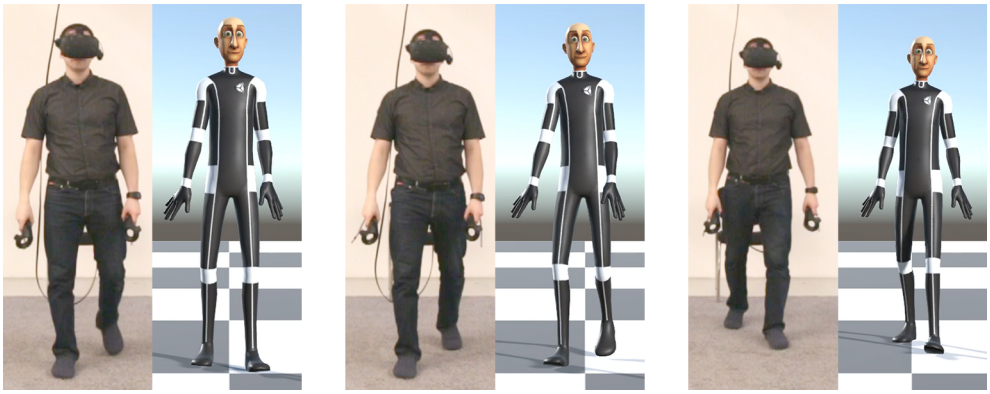

Figure 4.3. Detailed comparison of walking backwards

I also noted that the solution was little too responsive to the changes in head position while walking. A slightly drift from side to side can be observed in Figure 4.2. This sideways motion is not compensated in any way during the goal generation, which causes the character sometimes ending up slightly off center relative to the user input.

The Figure 4.3 also presents another issue with the solution responding to input. The first image-pair shows the user having taken a half a step backwards, without the solution responding in any way. This is due to the fact that the user's head stays relatively still during this time. And after the user's head starts moving, the solution is able to catch up to the user's motion, as can be seen in the later images.

4.5.3 Turning

Next, I evaluate how well the solution responded to turning.

Figure 4.4. Side-by-side comparison of turning motion

As Figure 4.4 shows, the solution responded well to changes in orientation, though some minor artifacts were noticeable.

Some of these artifacts can be seen in Figure 4.5. In the left-most image, the character and the user both stand still, but the character has a slightly offset from the desired position. The solution compensated this by having

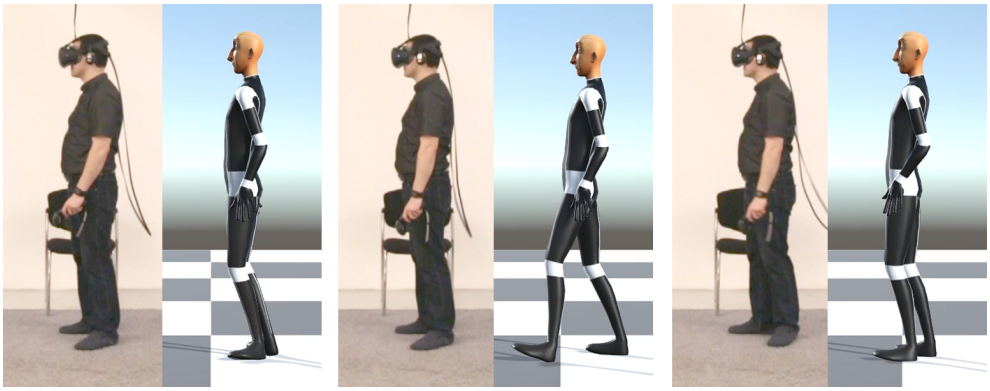

Figure 4.5. Artifacts in turning motion

the character take a corrective step forward (middle image). This issue relates to the previously mentioned way of extrapolating the trajectory from the HMD and the lack of compensation. The procedural touchups which related to the position adjustment would have diminished these artifacts, but with the cost of adding unnatural movement.

4.5.4 Sitting

Besides locomotion evaluated above, I wanted to see how the solution handled sitting and crouching motions. As both motions have relatively similar trajectories (head moving in a downward direction), I expected the solution to have some problems in determining the correct motion.

While the solution seems to produce a reasonable estimation, the character hesitated slightly (Figure 4.6). The hesitation is visible in the second image from the left, where the characters head position remains briefly in the original position.

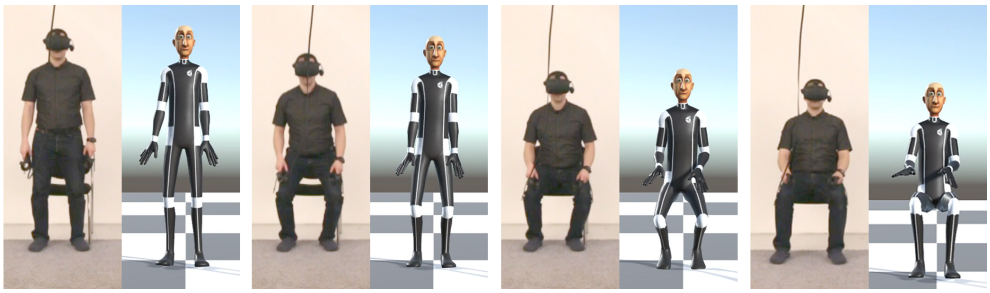

Figure 4.6. Side-by-side comparison of sitting motion

4.5.5 Crouching

Though crouching is not a motion which is easy to execute while wearing VR device, the solution did not seem to have any problems keeping up.

Moreover, it did not mix the crouching motion with sitting during the evaluation.

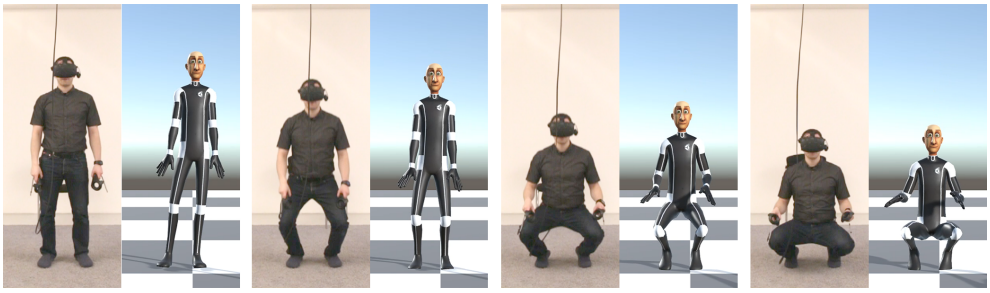

Figure 4.7. Side-by-side comparison of crouching motion

The character hesitated the same way as with sitting (Figure 4.7). I gathered that the cause of this hesitation is that there are multiple poses with the similar trajectory of downward motion in the pose library.

4.5.6 Hands

The IK solution provides an adequate approximation of the hand positions, and with an approximation of the local hand positions, they work pretty well when even when sitting and crouching. The most glaring issue with the hands seems to be the generation of unnatural wrist orientations (Figure 4.8).

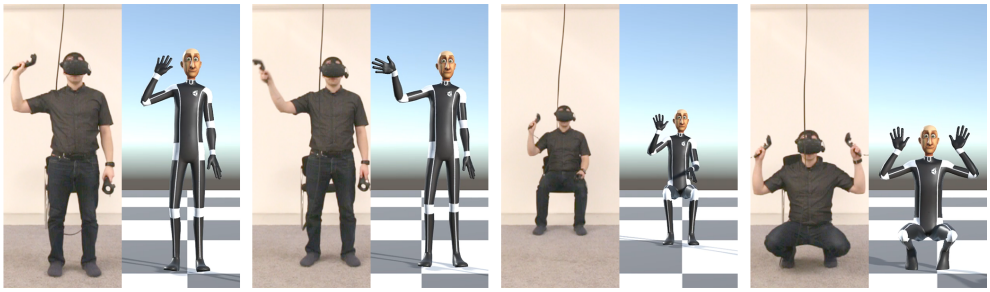

Figure 4.8. Side-by-side comparison of hand motions

5. Conclusions

This thesis began as an overview of the different data-driven animation methods, but in the thesis time grew to encompass the Full Body Problem, including a solution for it. This change ultimately affected the quality of the written part of the thesis. As the theoretical part only provides shallow computer animations and animation control techniques without. Also, solution's evaluation of the solution is done by the author.

Though the change in topic caused some issues, I still see it as a positive change. This is because I know that I produced useful knowledge when defining the Full Body Problem, which could be used to better frame character control problems. But the most critical and valuable outcome of this thesis was the solution, and how it showed a hybrid approach to the issue of controlling virtual characters using VR devices is a fruitful approach. This approach allowed the solution to map the tracked motion from the user almost directly to the character while animating the lower body with motion capture animations.

Additionally, I gained an understanding of the motion capture process from planning to importing the capture motions to Unity. This led me to create pipeline process to handle motion capture animations from raw data to skeleton animations. Also writing the thesis with LaTeX proved to a challenge on its own.

The solution produced motion for a virtual character which closely modeled the user input. Though there were multiple instances of motion artifacts, I'm satisfied with the overall results.

As an unexpected result, the solution has builtin flexibility toward the number of tracking data. The solution only requires the position and orientation of the head to produce motion, which is because the hybrid approach enables the motion capture animations to 'fill in' the missing parts. Conversely adding tracking points increases the accuracy of the

simulated motion due to the used IK solution, which offers FBIK support. The solution can be tailored to fit the input space of broad range of devices.

There is still research and development to be done for the solution. The motion artifacts discussed in the evaluation indicate that the solution requires some additional support for motion recognition to make better guesses from the user input. Also, support for a larger motion space should be investigated as the evaluation was done using a limited number of motions. Could the solution be extended to recognize kicking without additional tracking devices? This and many more questions might be answered as new methods for animation control are discovered.

Bibliography

Print Resources

Aristidou, A., Lasenby, J., Chrysanthou, Y., & Shamir, A. (2017). Inverse kinematics techniques in computer graphics: A survey. *Computer Graphics Forum*, *0*(0). doi:10.1111/cgf.13310. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13310

Bailenson, J. (2018). *Experience on demand: What virtual reality is, how it works, and what it can do*. New York, NY, USA: W.W. Norton & Company.

Herda, L., Fua, P., Plänkers, R., Boulic, R., & Thalmann, D. (2001). Using skeleton-based tracking to increase the reliability of optical motion capture. *Human movement science*, *20*(3), 313–341.

Holden, D., Komura, T., & Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Trans. Graph. 36*(4), 42:1–42:13. doi:10.1145/3072959.3073663

James, D. L. & Twigg, C. D. (2005). Skinning mesh animations. *ACM Trans. Graph. 24*(3), 399–407. doi:10.1145/1073204.1073206

Jiang, F., Yang, X., & Feng, L. (2016). Real-time full-body motion reconstruction and recognition for off-the-shelf vr devices. In *Proceedings of the 15th acm siggraph conference on virtual-reality continuum and its applications in industry - volume 1* (pp. 309–318). VRCAI '16. Zhuhai, China: ACM. doi:10.1145/3013971.3013987

Kovar, L., Gleicher, M., & Pighin, F. (2008). Motion graphs. In *Acm siggraph 2008 classes* (51:1–51:10). SIGGRAPH '08. Los Angeles, California: ACM. doi:10.1145/1401132.1401202

Parent, R. (2012). *Computer animation: Algorithms and techniques* (3rd). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

van de Panne, M. (2014). Motion fields for interactive character animation: Technical perspective. *Commun. ACM*, *57*(6), 100–100. doi:10.1145/2602759

Van Welbergen, H., Van Basten, B. J., Egges, A., Ruttkay, Z. M., & Overmars, M. H. (2010). Real time animation of virtual humans: A trade-off between naturalness and control. In *Computer graphics forum* (Vol. 29, *8*, pp. 2530–2554). Wiley Online Library.

Online Resources

Clavet, S. (2016a). Motion matching and the road to next-gen animation. Retrieved April 8, 2018, from http://www.gdcvault.com/play/1022985/Motion-Matching-and-The-Road

Clavet, S. (2016b). Motion matching for realistic animation in "for honor"(simon clavet). Retrieved April 8, 2018, from https://www.youtube.com/watch?v=4pdcA3mhe0E

Hofkamp, A. (2015). From user input to animations using state machines. Retrieved April 8, 2018, from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/from-user-input-to-animations-using-state-machines-r4155/

Mixamo.com. (2018). Mixamo animated 3d characters. Retrieved April 8, 2018, from https://www.mixamo.com

Mocappys.com. (2018). Mocappys - your guide to capturing and editing motion. Retrieved April 8, 2018, from http://mocappys.com/

Nikula, P. (2017). Uusi tuote vie työntekijät virtuaalitodellisuuteen. Retrieved April 8, 2018, from https://www.kauppalehti.fi/uutiset/uusi-tuote-vie-tyontekijat-virtuaalitodellisuuteen/mnANcwTs

Retargeting of Humanoid animations. (2018a). Retrieved April 8, 2018, from https://docs.unity3d.com/Manual/Retargeting.html

State Machine Transitions. (2018b). Retrieved April 8, 2018, from https://docs.unity3d.com/Manual/StateMachineTransitions.html

Raw Mocap Data for Mecanim. (2012). Retrieved April 8, 2018, from https://assetstore.unity.com/packages/3d/animations/raw-mocap-data-for-mecanim-5330

University, A. (2012). T-talo. Retrieved April 8, 2018, from http://media.tkk.fi/fi/laitos/tilat-valineet/t-talo/